Constructing Orbit Integrators in Python: Observing Energy Conservation and Comparison to Stellar Stream Orbits

Tom Holmes
Department of Physics, University of Surrey, Guildford, GU2 7XH, UK

The construction and use of integrators is becoming prevalent in the study and modelling of orbits seen on galactic bodies, such as stellar streams and globular clusters. The two integrators tested in this work are the Euler method and the Kick-Drift-Kick (or 'Leapfrog') method. These models compute the location, velocity, and acceleration of the next step in an orbit with basic parameters of the central body and the current position and velocity. The more accurate an orbit integrator, the more efficient it is at conserving potential energy throughout the orbit cycle. The Euler method, the simpler of the two, proved to lose energy throughout its orbit, while the Leapfrog method conserved its energy almost completely. The KDK method can be further perfected with the inclusion of variable timesteps and an eccentric launch velocity, to produce an ellipse. This orbit is accurate and resembles the movements seen in measured astrophysical data. Using this KDK variable timestep model, mock data of stellar stream orbits can be plotted against the simulation, which can then be tailored to match the data, finding values for orbit energy and central body mass. Potential further research would be to incorporate a multi-body central mass, and observing how the orbit would vary with the mass and distance between these bodies.

I. INTRODUCTION & THEORY

As the research field of cosmology and galaxy evolution grows, the desire to understand the movement, orbits and patterns we see throughout a range of galactic bodies, expands. Astronomical data has been gathered evermore frequently over the past 3 decades. Telescopes such as the Hubble Space Telescope (HST) [1] and astronomical surveys like the Sloan Digital Sky Survey (SDSS) [2] paved the way for a new era of observational astronomy. There are new telescopes on the horizon that also hold promising futures such as James Webb Space Telescope (JWST) [3] and ARRAKIHS [4]. However, to accompany this observational data, a strong base and field of computational astronomy has been formed, with a branch dedicated to mapping and modelling the orbits of galaxies, stellar streams and globular clusters. Plotting and understanding these movements, allows us to gain insight into the properties of the bodies and their patterns over time. A poignant example of this is the research into the total masses of galaxies (baryonic matter + dark matter). In 2019, the total mass of the Large Magellanic Cloud (LMC) was calculated due to it's effect on the orbit of a stellar stream, called the Orphan Stream [5]. This was possible due to the mapping and calculation of the LMC orbit using orbit integrators. The integrator algorithm used in this research prior mentioned is of multiple orders, with complex dynamics, and is not the focus of this work. Instead, the core principles of what constructs an accurate, repeatable and efficient orbit integrator is what will be discussed along with further testing and applications.

A. Keplerian Mechanics

In galactic dynamics, the acceleration of a body in orbit, with negligible mass, is given by equation [1]

$$\ddot{\vec{x}} = -\vec{\nabla}\phi(\vec{x}) \qquad [1]$$

If the orbiting body is considered to be Keplerian and orbiting around a central body of mass, M, the Keplerian potential is

$$\phi(\vec{x}) = -\frac{GM}{r} \qquad [2]$$

Where r is the magnitude of the position vector

$$r = \sqrt{x^2 + y^2 + z^2} \qquad [3]$$

Therefore, the acceleration for a Keplerian potential can be shown as

$$\ddot{\vec{x}} = -\frac{GM}{r^3}\vec{x} \qquad [4]$$

With these equations, they can be integrated over small time intervals in which the acceleration is believed to be constant, allowing for the position and velocity vectors to be found in the sequential timestep. We will explore 2 methods of this integration: Euler and Kick-Drift-Kick (KDK), the latter being an example of a Leapfrog integrator. A suitable comparison to make between models is the energy conservation throughout the orbit. The total energy of the orbit is given in equation [5]:

$$E = \frac{1}{2}v^2 - \frac{GM}{r} \qquad [5]$$

The first term in this equation describes the kinetic energy, with the latter being the potential energy (in a Keplerian potential). Comparing the energy of the orbit integrators provides a convenient way of demonstrating the accuracy of the models. An orbit should conserve it's energy perfectly, transferring potential for kinetic throughout it's journey. This conservation allows it to maintain a stable path, not spiralling inwards or outwards if there is too much potential or kinetic energy.

B. Euler Method

The Euler Method is a First-Order Integrator, and is named after Leonhard Euler who formulated it in 1768 [6]. The integrator takes position ($\overrightarrow{x_n}$) and velocity ($\overrightarrow{v_n}$) vectors of an object at time, t = n. Over a defined timestep, the sequential position ($\overrightarrow{x_{n+1}}$) and velocity ($\overrightarrow{v_{n+1}}$) vectors at time, t = n+1 can be calculated using equations [6] and [7].

$$\overrightarrow{x_{n+1}} = \overrightarrow{x_n} + \overrightarrow{v_n}\Delta t \qquad [6]$$

$$\overrightarrow{v_{n+1}} = \overrightarrow{v_n} + \overrightarrow{a_n}\Delta \mathrm{t} \qquad [7]$$

Δt is the timestep at which the following vectors will be calculated. $\overrightarrow{a_n}$ is the acceleration vector at time, t = n. Equation 4 written in accordance with this notation is

$$\overrightarrow{a_n} = -\frac{GM}{[\overrightarrow{x_n}]^3} \times \overrightarrow{x_n} \qquad [8]$$

G is the gravitational constant (43007.10573 $\frac{kpc}{M_\odot(\frac{km}{s})^2}$) and M is the mass of the central body that our object is orbiting around.

C. Kick-Drift-Kick (Leapfrog) Method

Where the Euler method was a First-Order Integrator, the Kick-Drift-Kick (KDK) model is a Second-Order integrator, with an additional step included in the position and velocity calculations. The Euler method computes both parameters across the same timestep, whereas KDK has a half timestep that splits up the calculations. Velocity vector is calculated over half a timestep (Kick) [equation 9], the position vector is then computed over a full timestep (Drift) [equation 10], and finally the velocity vector is calculated for the second half of the timestep (Kick) [equation 11]. KDK is known as a

Leapfrog algorithm. Drift-Kick-Drift (DKD) is another example of a Leapfrog algorithm, with the KDK steps reversed.

$$\overrightarrow{v_{n+\frac{1}{2}}} = \overrightarrow{v_n} + \overrightarrow{a_n}\frac{\Delta t}{2} \qquad [9]$$

$$\overrightarrow{x_{n+1}} = \overrightarrow{x_n} + \overrightarrow{v_{n+\frac{1}{2}}}\Delta t \qquad [10]$$

$$\overrightarrow{v_{n+1}} = \overrightarrow{v_{n+\frac{1}{2}}} + \overrightarrow{a_{n+1}}\frac{\Delta t}{2} \qquad [11]$$

D. KDK: Variable Timesteps

Both versions of the Euler and KDK integrators discussed involve constant timesteps. For a circular orbit, this is acceptable, however real data show that orbits are almost always eccentric. This means that the acceleration near the pericentre and apocentre of the orbit is vastly greater than that seen in the 'flatter' sections of the orbit. For algorithms with many orbits, computing each orbit with constant timesteps can be very slow and costly, as vector calculations are taken as frequently in the slower regions as they are in the faster areas (pericentre/apocentre). A method was derived that produced an eccentric orbit and kept computing costs and time down. Using a dimensionless parameter, η (eta), the timestep for each step in the integrator can be based upon the acceleration and position of the object at a specific point in it's orbit.

$$\Delta t = \eta\sqrt{\frac{r_n}{a_n}} \qquad [12]$$

In equation [12], the magnitude of the timestep has an inverse-squared relation with acceleration. If the acceleration is larger (e.g. at pericentre/apocentre), the timestep will be much smaller, or shorter. This allows for more data to be computed at these vital points in the orbit, whereas the slower, less active parts have slightly longer timesteps, saving on price and time. The launch velocity can also be adapted to incorporate apocentre and pericentre positions and velocities. Equation [13] shows the calculation for v_{apo} which is the launch velocity.

$$v_{apo} = \sqrt{\frac{2(\frac{GM}{r_{peri}} - \frac{GM}{r_{apo}})}{\left(\frac{r_{apo}}{r_{peri}}\right) - 1}} \qquad [13]$$

This is derived from conservation of energy and angular momentum. See Appendix A for full derivation.

This method also allows for the orbit to be eccentric due to this inequality of timesteps at various points in the orbit. There isn't a 'perfect' value for η, but it can be varied to find a compromise between the number of desired data points, and the computational run-time, and therefore price.

II. COMPUTATIONAL DETAILS

Python was used in the Spyder Software to test and evaluate these orbit integrators. First, core parameters needed for the calculations were imported or defined.

- Gravitational constant = 43007.10573 (units of $\frac{kpc}{M_\odot(\frac{km}{s})^2}$)
- Mass of central body = 9 (units of $\times\ 10^{10}\ M_\odot$)

The value of M was provided by the University of Surrey as an estimate of the mass of the Milky Way encompassed within the radius of the Sun.

Functions for the integrators were constructed, taking the initial position and velocity vectors, and the timestep at which the calculations were repeated. These functions computed the following position and velocity vectors and returned them. The KDK integrator also took the initial acceleration as an argument as equation [11] incorporates the acceleration at the time n+1. See Appendix B for the Euler and KDK (constant and variable timesteps) functions in Python code.

These functions were then iterated via 'while' loops, appending to coordinate and velocity lists to compile datasets that can later be plotted and analysed. The use of a 'while' loop was beneficial as it allowed iterations to be computed until a certain condition was met, with said condition being an 'end' time value, such as 5 Gyr. At the end of these 'while' loops, a time array was amended to add the timesteps for each iteration. This allows for a time dimension to be defined, for analysis and plotting.

Once the coordinate and velocity lists were complete for Euler and KDK (constant and variable timesteps), the energy could be computed as a function of time, using equation [5]. 'For' loops were used in these calculations, taking the position vector and velocity vector magnitude at index [i]. The energy was then stored as a list for each integrator.

The packages used in the code were NumPy [7] and matplotlib.pyplot [8]. NumPy had functions such as linalg. norm() which takes an array as it's argument, computing it's magnitude, or r (equation [3]). Matplotlib.pyplot was utilised to construct plots that will be shown within this report.

The uncertainties of input parameters were not taken as they are negligible and would provide no insight or relevance in the orbit. However, energy and its uncertainty were calculated for each simulation as all methods conserve energy in different ways. The energy for the simulations is found via for loops, creating a list of energy values for each timestep throughout the orbit. Uncertainties were found through the standard deviations of these lists and printed as a percentage error.

III. RESULTS AND DISCUSSION

The Euler method was the initial integrator to be constructed and evaluated. The position vector, $\vec{x}$, was based upon the distance of the Sun to the centre of the Milky Way, 8 kpc. In these models, the velocity, $\vec{v}$, was launched in the positive y-direction, calculated via $\vec{v} = \sqrt{\frac{GM}{r_0}}$, $\vec{v} = 219.96\ \text{km}s^{-1}$.
The Euler method has a spiral-like appearance and does not conserve radius or energy, Figure [1]. It shows the failure of energy conservation from a first-order integrator, such as Euler. The Euler Integrator can delay its spiral outwards if the timestep duration is shortened. This does, however, only delay the inevitable, with the orbit eventually spiralling outwards. However, over a relatively short orbit period (e.g., $\leq$3 Gyr), a more conservative orbit can be obtained if the timesteps are shortened substantially, Figure [2].

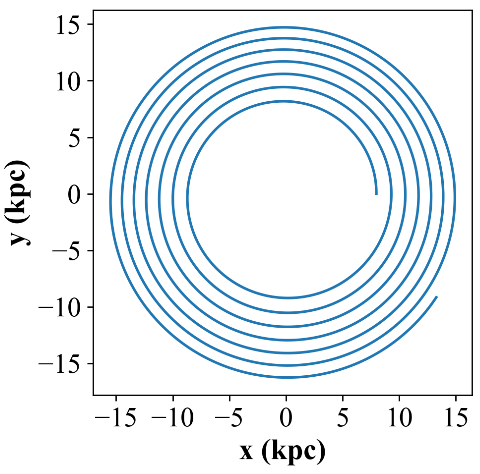

FIG. 1: Plot of the X,Y coordinate space for Euler Integrator Method, with starting conditions: $\vec{x}$ (t=0) = [8, 0, 0] kpc, $\vec{v}$ (t=0) = [0, 219.96, 0] kms^{-1}, Δt = 0.0005 (kpc/kms^{-1} or Gyr). This orbit has an elapsed time of 3 Gyr.

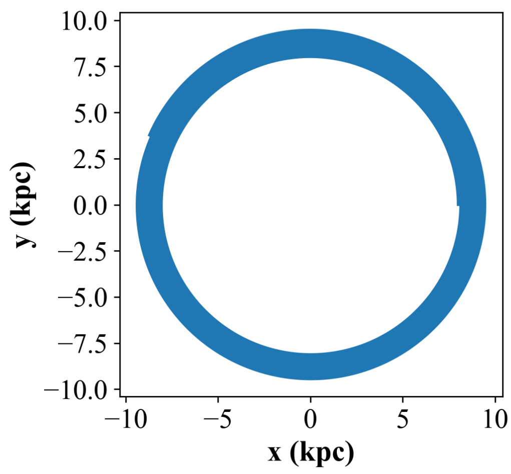

FIG. 2: Plot of the X,Y coordinate space for Euler Integrator Method, with starting conditions: $\vec{x}$ (t=0) = [8, 0, 0] kpc, $\vec{v}$ (t=0) = [0, 219.96, 0] kms^{-1}, Δt = 0.00005 (kpc/kms^{-1} or Gyr). This orbit has an elapsed time of 3 Gyr.

The KDK integrator constructed an orbit with identical starting position and velocity as Euler. To conduct a direct comparison, the KDK timestep will be double the timestep for Euler due to the additional step seen in the KDK integrator, Figure [3]

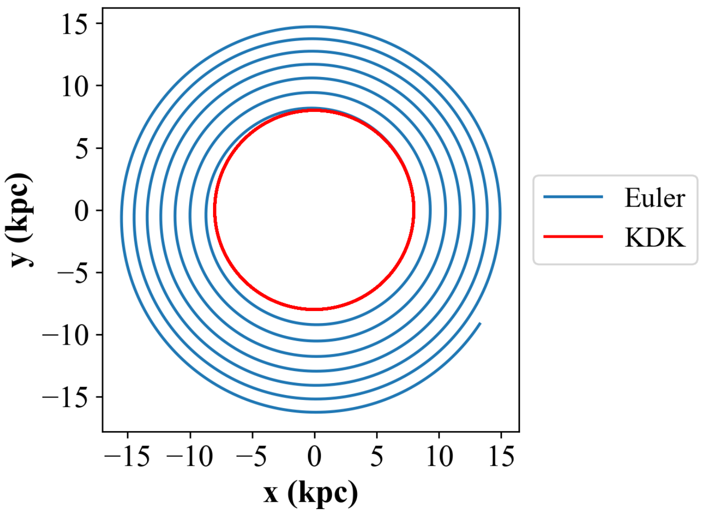

FIG. 3: Plot of the X,Y coordinate space for Euler Integrator (blue) and KDK Integrator (red) Methods, with starting conditions: $\vec{x}$ (t=0) = [8, 0, 0] kpc, $\vec{v}$ (t=0) = [0, 219.96, 0] kms^{-1}. For Euler, Δt = 0.0005 (kpc/kms^{-1} or Gyr). For KDK, Δt = 0.001. Both orbits have an elapsed time of 3 Gyr.

In this direct comparison, it is clear to see that KDK does not spiral out like Euler. The integration steps do not allow for this spiralling behaviour to occur, constructing a much more stable orbit that can be ran for much longer periods, with no concern regarding loss of energy. KDK does display minor fluctuations of radius throughout the orbit that are not visible in Figure [3], but these fluctuations do not have a ‘drift’ element, and constantly oscillate.

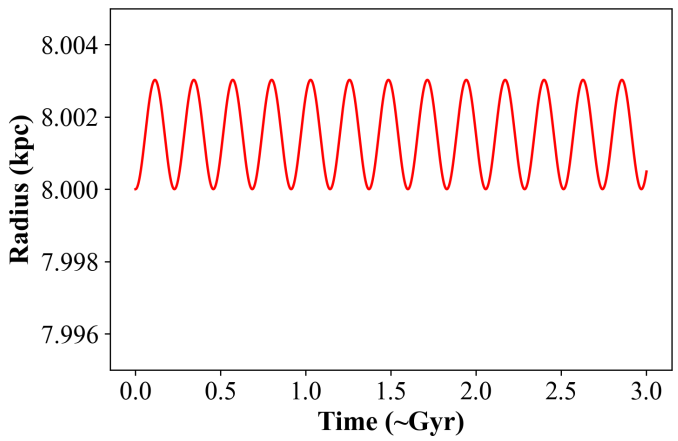

FIG. 4: Plot of the radius throughout the KDK orbit seen in Figure [3]. The starting radius is 8 kpc, and throughout the entire duration of the period, it fluctuates with a period of 0.23 $\pm$ 0.01 Gyr. There is no 'drift' feature of this movement, allowing the radius to be assumed constant, and therefore conserved.

This conservation of radius in the KDK model, also insinuates its conservation of energy. Euler does not demonstrate the same conservation of radius as KDK and therefore, it does not conserve energy throughout the orbit. How quickly this energy is lost depends on the timesteps used for the integration, with longer timesteps demonstrating a much more rapid loss of radius and energy in the Euler model. Figure [5] shows the energy loss for the orbits seen in Figure [3].

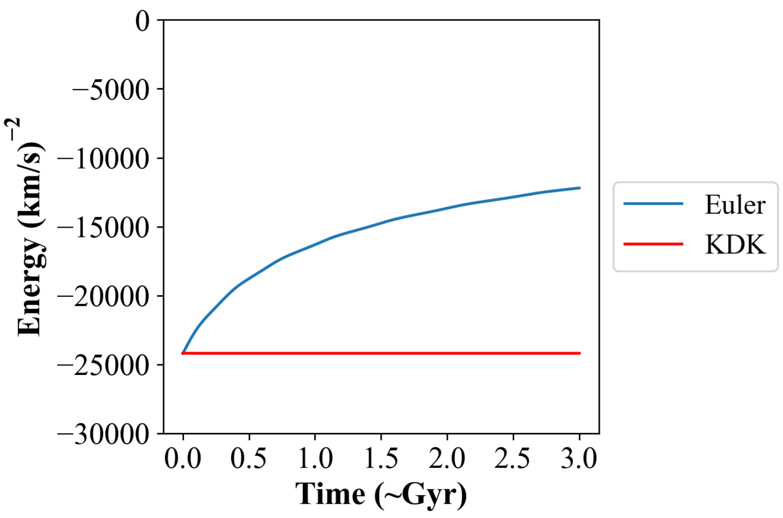

FIG. 5: Plot of the energy conservation over time for Euler Integrator (blue) and KDK Integrator (red) Methods with orbits seen in Figure [3]. KDK has a very stable total energy, for this orbit it is -24191.497 $\pm$ 0.001 (km/s)$^{-2}$. The rate of energy loss for Euler varies depending on Δt. Δt = 0.0005 for Euler in this plot.

KDK is substantially more accurate and stable than Euler, however with constant timesteps it still constructs a circular orbit, whereas in a Keplerian potential, we expect to see eccentric orbits. To enable this, variable timesteps can be utilised instead of the constant values used thus far, and the launch velocity calculation will be modified to accommodate a set apocentre and pericentre. As seen in equation [12], variable timesteps are calculated via the radius and acceleration of the object and will be used in the calculations to compute the new position and velocity. More points will be computed at the high acceleration areas, producing an ellipse. To demonstrate the effectiveness of this, η = 0.003. This value was found to be a suitable balance between computation time and a stable orbit.

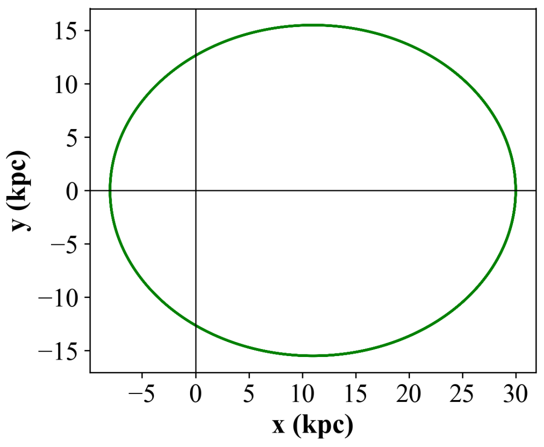

FIG. 6: Plot of the X,Y coordinate space for KDK integrator with variable timesteps, with $\eta = 0.003$. For this simulation, the starting conditions are modified to accommodate for the eccentricity. A set of axes are positioned at the origin to assist the visualisation of the new elliptical properties. The starting position is still [8, 0, 0] (apocentre), but the velocity is calculated to accommodate for a pericentre of [-1, 0, 0].

This eccentric orbit is far more accurate to orbits seen in astronomical data, from planets to stellar streams. As η increases, the uncertainty in the radius increases, with a thicker orbit curve being constructed, indicating a wider amplitude of radius fluctuations. For $\eta \leq 0.02$, there isn't a discernible increase in precision that warrants the longer processing time needed for small values of η. From constant to variable timesteps, the energy conservation in KDK is strong throughout and is not a reason to consider the use of the variable method.

Using mock stellar stream data prepared by Dr Denis Erkal at the University of Surrey, the KDK variable timestep model can be used to compare against this mock data. To match the parameters of the stream, the starting position (apocentre) is [4.979649, -14.694955, 0] and the launch velocity is [165.772981, -45.175504, 0].

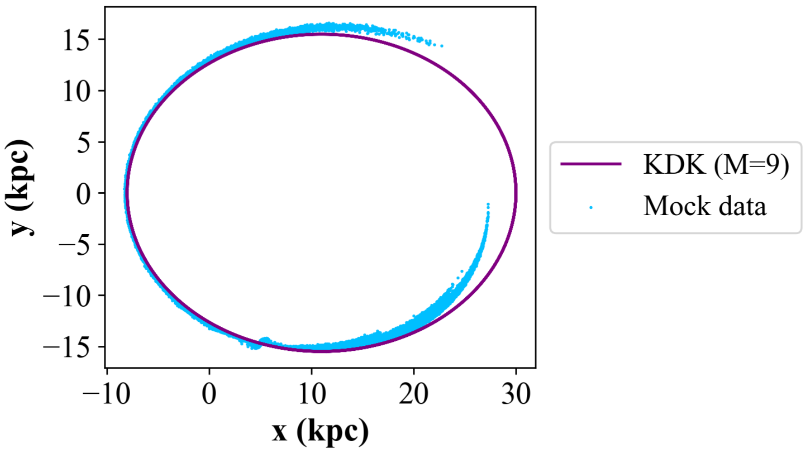

FIG. 7: Plot of the X,Y coordinate space of KDK variable timestep model mapped against stellar stream orbit mock data provided by the University of Surrey Physics Department. $\eta = 0.003$ in this simulation. The central body for this orbit contains the same mass used throughout the simulations, $9 \times 10^{10} M_{\odot}$.

The energy of the KDK model with these parameters is -10185.760 $\pm$ 0.017 (km/s)$^{-2}$. The leading arm, seen around [25, -10], seems to contain a lower energy due to its motion of falling towards the central body, whereas the trailing arm, around [15, 15], appears to feature a greater energy, causing this dispersing behaviour, with stars trailing off from the stable orbit. If we vary the central mass to achieve orbit simulations that align with these trailing and leading arms, we can discover their average energies. To match the leading arm of the stellar stream data, the central mass used in the simulation was set as $9.45 \times 10^{10}\ M_{\odot}$. This simulation computed the energy as -10036.9 $\pm$ 640.4 (km/s)$^{-2}$. For the trailing arm, the central mass was set as $8.73 \times 10^{10}\ M_{\odot}$, and the energy calculated was -10223.7 $\pm$ 364.0 (km/s)$^{-2}$. Figure 8 shows these orbit simulations against the mock data.

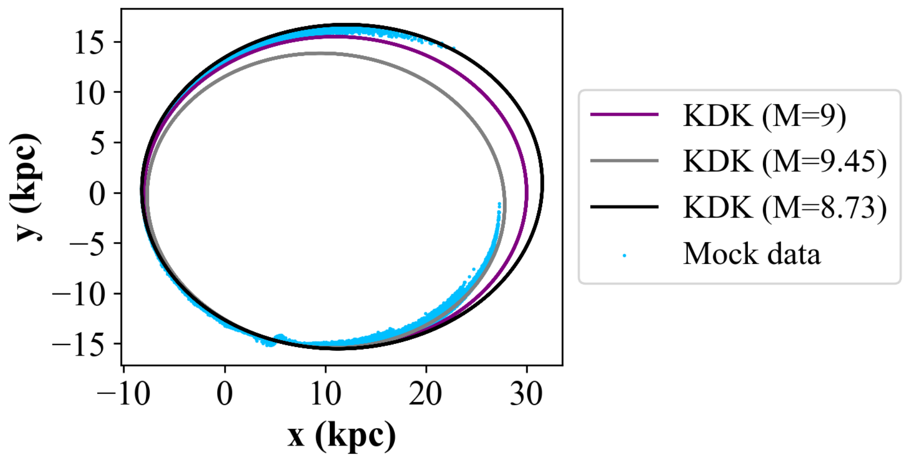

FIG. 8: Plot of the X,Y coordinate space of numerous KDK variable simulations with different central body masses to align with various parts of the stellar stream mock data. These simulations have masses of 9, 9.45, $8.73 \times 10^{10}\ M_{\odot}$. If the central mass of an orbit of measured data is unknown, mapping these orbit simulations with varying mass can be extremely useful to find the true value for the data.

IV. CONCLUSION

A Keplerian potential should produce an ellipse, however the Euler method calculates a circular orbit, that does not conserve energy. It is a simple and quick method that could be suitable for modelling known circular orbits with small orbital periods, accommodating for very short timesteps to delay the loss of energy. It does require a small amount of processing power and time, making it desirable for fast deployment of orbit modelling. However, it is not repeatable and not accurate for real orbits and should not be utilised given the alternate options available, such as Leapfrog integrators, such as KDK, that are far more accurate. KDK achieved energy conservation but was not accurate until the launch velocity and timestep length were adapted. These changes allowed for an eccentric orbit to be achieved, proving to be accurate to stellar orbit mock data. The mock data used was accurate to true orbits seen in astrophysical data, featuring trailing and leading arms that are influenced by other bodies such as dwarf galaxies and other streams. The KDK model parameters can quickly be varied to plot against stream data, allowing for analysis on properties such as the central body mass, which in the orbits discussed, is the central mass of the Milky Way. It also provides insight into the energy of the orbits, allowing for further insight into the energy changes throughout measured orbit data.

The simulations discussed are not of likeness to cutting edge research models, which involve further analytical and numerical methods, modelling millions of object orbits and their interactions. The KDK variable timestep model is a good approximation for stellar streams and provides great insight into galactic bodies and their properties. However, further study could be conducted to see if orbits of other bodies, not just stellar streams, can be appropriately mapped with the simulation. With more research time, developing a multi-body algorithm would also be of great value, analysing how orbit movements change with numerous bodies interacting with the orbiting body. If the mass of one body is known, mapping the orbit could potentially allow for the calculation of the parameters of the other body.

APPENDIX A

Derivation of v_{apo} for eccentric launch velocity

Conservation of total energy throughout orbit:

$$\frac{1}{2}v_{apo}^2 - \frac{GM}{r_{apo}} = \frac{1}{2}v_{peri}^2 - \frac{GM}{r_{peri}}$$

Conservation of angular momentum throughout orbit:

$$r_{apo}v_{apo} = r_{peri}v_{peri}$$

$$v_{peri} = \frac{r_{apo}v_{apo}}{r_{peri}}$$

$$\frac{1}{2}v_{apo}^2 - \frac{GM}{r_{apo}} = \frac{1}{2}(\frac{r_{apo}v_{apo}}{r_{peri}})^2 - \frac{GM}{r_{peri}}$$

$$\frac{1}{2}v_{apo}^2(\left(\frac{r_{apo}}{r_{peri}}\right)^2 - 1) = \frac{GM}{r_{peri}} - \frac{GM}{r_{apo}}$$

$$v_{apo} = \sqrt{\frac{2(\frac{GM}{r_{peri}} - \frac{GM}{r_{apo}})}{\left(\frac{r_{apo}}{r_{peri}}\right) - 1}}$$

Where M, r_{apo} and r_{peri} are user-defined coordinates, and G is the gravitational constant.

APPENDIX B

Python Functions for Euler and KDK (Constant and Variable) Integrators

```
#------------------------------------------------------------------------------------------------------------

#Defining Euler Step function
def Euler_step(location_0, v_0, delta_t):

    a = (-G*M/(np.linalg.norm(location_0)**3))*location_0
    location_new = np.add(location_0,(v_0*delta_t)) #Creating next position
    v_new = np.add(v_0, (a*delta_t)) #Creating next velocity

    return location_new, v_new

#------------------------------------------------------------------------------------------------------------

a_0 = (-G*M/(np.linalg.norm(location_0)**3))*location_0
```

```
#Defining Leapfrog function
def Leapfrog_step(location_0, v_0, a_0, delta_t):

    v_halfstep = np.add(v_0,(a_0*(delta_t/2)))
    location_new = np.add(location_0,(v_halfstep*delta_t))
    a_new = (-G*M/(np.linalg.norm(location_new)**3))*location_new
    v_new = np.add(v_halfstep,(a_new*(delta_t/2)))

    return location_new, v_new, a_new

#-----------------------------------------------------------------------------------------------------------------

# Defining Leapfrog integrator with variable timesteps
def Leapfrog_step_variable_t(location_0, v_0, a_0):

    eta = 0.003
    vdelta_t = eta*np.sqrt(np.linalg.norm(location_0)/np.linalg.norm(a_0))
    v_halfstep = np.add(v_0,(a_0*(vdelta_t/2)))
    location_new = np.add(location_0,(v_halfstep*vdelta_t))
    a_new = (-G*M/(np.linalg.norm(location_new)**3))*location_new
    v_new = np.add(v_halfstep,(a_new*(vdelta_t/2)))

    return location_new, v_new, a_new,eta, vdelta_t

#-----------------------------------------------------------------------------------------------------------------
```

REFERENCES

[1] NASA HubbleSite (https://hubblesite.org/home) (accessed 17/10/2023)
[2] SDSS (https://www.sdss.org/) (accessed 17/10/2023)
[3] NASA James Webb Space Telescope (https://webb.nasa.gov/) (accessed 17/10/2023)
[4] ESA ARRAKIHS FactSheet (https://www.esa.int/Science_Exploration/Space_Science/Arrakihs_factsheet) (accessed 17/10/2023)
[5] D. Erkal et al. (2019) *The total mass of the of the Large Magellanic Cloud from its perturbation on the Orphan Stream*, Monthly Notices of the Royal Astronomical Society: Letters **487** (Pages 2685-2700)
[6] L. Euler (1768-1770) *Institutiones Calculi Integralis*
[7] NumPy Documentation (https://numpy.org/) (accessed 26/10/2023)
[8] MatPlotLib Documentation (https://matplotlib.org/) (accessed 26/10/2023)